Create PDF with Markdown

Marcus Haupt
marcus@haupt.contact
https://marcus.haupt.contact

About this documentation

This Note is about the procedure to create beautiful PDFs from markdown files managed by Dendron. Dendron does not offer an option to export the Markdown files as PDF. Therefore I use the two additional tools Pandoc and Weasyprint.

Pandoc is a universal document converter that can convert Markdown files to many other file types. With Pandoc alone, it is already possible to generate PDFs from Markdown files. However, Weasyprint offers an easy to use and at the same time extensive possibility to design the PDFs beautifully with CSS.

Weasyprint on the other hand can only work with HTML files. Therefore I need both tools for the workflow. First, an HTML file is generated with Pandoc, which is then formatted into a nice PDF with Weasyprint and a CSS template.

Use Pandoc to convert Markdown files to HTML files

To use Pandoc, the tool must first be installed. The installation routine depends on the operating system and the architecture used. The general information can be found on the official website (Installing Pandoc).

I am using a Macbook Pro M1 Max at the time of documentation. Fortunately, the package manager Homebrew offers the package from its repository.

```
brew install pandoc
```

For the pure conversion of Markdown files to HTML files no add-on is needed. I just create a HTML template that Pandoc should follow. The HTML template comes from the blog of Arthur Kiezel and was only slightly adapted. In his blog Arthur explains the conversion process. The blog article can be read here https://arthurkoziel.com/convert-md-to-html-pandoc/.

```
 <!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="date" content='$date-meta$'>
    <title>$title$</title>
  </head>
  <body>
    <header></header>
    <h1>$title$</h1>
    $body$
    <footer></footer>
  </body>
</html>
```

I use strictly the directory structure of Dendron for this. All files that do not refer to Markdown are stored in the directory `assets`. Here I have created a directory structure for all file types involved in this process. I highly recommend to build such a structure for a clean process.

```
 % tree -L 3
.
├── assets
│   ├── docs
│   │   ├── html
│   │   └── pdf
│   └── templates
│       ├── css
│       ├── fonts
│       └── html
├── create-pdf-with-markdown.md
├── root.md
└── root.schema.yml
```

To convert a Markdown file, the source file and the HTML template.

```
% pandoc --standalone --template assets/html/template.html create-pdf-with-markdown.md
```

Pandoc assumes the file format from the file ending and converts the file to HTML from default. Per default Pandoc just sends the converssion to `STDOUT`. To save it as a HTML file we simply send it to a specified file.

```
% pandoc --standalone --template assets/html/template.html create-pdf-with-markdown.md > assets/html/
create-pdf-with-markdown.html
```

You can find the html file with this link:

create-pdf-with-markdown

The HTML template takes some of the metadata from the Markdown file like the title and the content (body). The resulting HTML file has no formatting and is therefore well suited for further conversion with Weasyprint.

Use Weasyprint to convert HTML files to PDFs

After a flat HTML file is available, a beautiful PDF can be created from it with the help of Weasyprint. Here I would like to point out that there are hardly any limits to the design. I like to keep my PDFs as unagitated as possible, so I focus primarily on proper formatting and readability. Weasypringt offers impressive examples for reports, tickets and books on their website.

The most important step is to create a CSS template that will be used to format the HTML file in PDF. Here I pay attention above all to the fact that it corresponds to my local standard. Since I live in Germany, I set the styles all based on the DIN A4 standard. Below is my standard template that I use as a base for most of my documents.

```
 /* ================================
   Font Styles
================================ */

@font-face {
  font-family: IBM Plex Sans;
  font-weight: 400;
  src: url(../fonts/IBMPlexSans-Regular.ttf);
}
 @font-face {
  font-family: IBM Plex Sans;
  font-style: italic;
  font-weight: 400;
  src: url(../fonts/IBMPlexSans-Italic.ttf);
}
@font-face {
  font-family: IBM Plex Sans;
  font-weight: 300;
  src: url(../fonts/IBMPlexSans-Light.ttf);
}
@font-face {
  font-family: IBM Plex Sans;
  font-style: italic;
  font-weight: 300;
  src: url(../fonts/IBMPlexSans-LightItalic.ttf);
}
@font-face {
  font-family: IBM Plex Sans;
  font-weight: 700;
  src: url(../fonts/IBMPlexSans-Bold.ttf);
}
@font-face {
  font-family: IBM Plex Mono;
  font-weight: 400;
  src: url(../fonts//IBMPlexMono-Regular.ttf);
}

/* ================================
   Page Styles
================================= */

@media print {
  @page {
    size: A4;
    margin: 2.5cm 1.5cm 2.5cm 1.5cm;
    @top-left {
      content: "IBM Technology Lificycle Support";
      font-size: 9pt;
      font-weight: bold;
      height: 1cm;
      text-align: left;
      width: 10cm;
    }
    @top-center {
      background: #0f62fe;
      content: '';
      display: block;
      height: .05cm;
      opacity: .5;
      width: 100%;
    }
    @top-right {
      content: string(heading);
      font-size: 9pt;
      height: 1cm;
      vertical-align: middle;
      width: 100%;
    }
    @bottom-left {
        content: "IBM non-offcial documentation. Internal project use only. Created by Marcus Haupt.";
        font-size: 9pt;
        height: 1cm;
        text-align: left;
        width: 9cm;
    }
    @bottom-center {
        background: #0f62fe;
        content: '';
        display: block;
        height: .05cm;
        opacity: .5;
        width: 100%;
      }
    @bottom-right {
        font-size: 9pt;
        content: counter(page);
        height: 1cm;
        text-align: right;
        width: 0.5cm;
    }
  }
}

/* =================================
   HTML Classes
================================= */
```

```
html {
  color: #393939;
  font-family: IBM Plex Sans;
  font-size: 11pt;
  font-weight: 300;
  line-height: 1.5;
}

h1 {
  color: #0f62fe;
  font-size: 22pt;
  width: 100%;
}
h2, h3, h4 {
  color: black;
  font-weight: 400;
}
h2 {
  break-before: always;
  font-size: 20pt;
  string-set: heading content();
}
h3 {
  font-weight: 300;
  font-size: 15pt;
}
h4 {
  font-size: 13pt;
}

code {
  font-family: IBM Plex Mono;
  font-weight: bold;
}

/* ================================
   Custom Classes
================================ */

.sourceCode {
  background: #393939;
  color: aliceblue;
  font-family: IBM Plex Mono;
  font-weight: 400;
  margin-top: 5pt;
  padding-left: 2pt;
  padding-right: 2pt;
}

.sourceCode pre {
  white-space: pre-wrap; /* Allow line breaks within words */
  word-wrap: break-word; /* Break words when they reach the container edge */
  font-size: 8pt; /* Adjust font size as needed */
}

/* ================================
   End of Styles
================================ */
```

For the creation of the CSS template I followed this very good video from the company clickbar.. The video can be found on YouTube at From HTML to PDF with WeasyPrint. Note: the video is in German.

In addtion to that video I used the examples provided by Weasyprint on their official GitHub repository. You can found it here: Weasyprint Examples.

Once the CSS template is in place, it is again just a command that needs to be executed for conversion.

```
 % weasyprint assets/html/create-pdf-with-markdown.html assets/pdf/create-pdf-with-markdown.pdf -s
assets/css/default-styles.css
```

After that you will have a nicely formatted PDF document ready to send or print. You can find the document from this example here: PDF download.